34 | 容量规划：如何精准地对生产系统做容量预估？

2022-01-04 高楼

《全链路压测实战30讲》 课程介绍 >

讲述：高楼

时长 20:16 大小 18.56M

你好，我是高楼。

到这里啊，我们专栏的主体内容已经讲得差不多了，场景也已经跑起来了。这节课，我们就来聊一聊容量规划。

对于一个线上系统来说，精准地判断系统最大容量是保证系统能稳定运行的前提条件，而对于大部分系统来说，它们的规划和评估都太过粗略，以至于不是浪费了资源就是随时等死，所以容量评估也必然是性能项目要产出的结论之一。这也是全链路线上压测出现的原因。

对于系统级的容量评估，在当前的技术市场中，除了拿硬件怼上去之外，似乎很难找到什么有效的方法。

我知道有些人会说，不是有 TPC-C、TPC-E...之类的吗？不是有基于 CPU 算力的 XX 模型吗？不管你怎么说，在实际的工作场景中，大部分的算法也还是下面这样的：

> 当一个节点支持 100TPS 时，有 100 个同样的节点就大概能支持:
> 100 × 100 × 80% ≈ 8000（TPS）

之所以乘以个 80%，是因为我们从心理上就觉得，实际场景可以支持的 TPS 总量好像可以，但又大概率达不到线性增加的程度，因为一定会有性能的损耗。但是这样算来显得也太不专业了，一个复杂的系统怎么能用小学生就能理解的公式来表达呢。确实，这么简单的公式会忽略很多问题，它过于简单粗暴不说，也非常不精准。

所以，一些架构师在做系统规划的时候会根据 TPC-C 来计算，下面我们就来说一说这种计算方式，同时也聊一聊磁盘和网络容量的评估方式。

TPC-C 容量评估

在讲解 TPC-C 之前，我们先来了解一下 TPC。TPC（Transaction Processing Performance Council）的全称是事务处理性能委员会，它是一个由数十家会员企业组成的非盈利组织。TPC 主要负责制定商务应用基准性能测试的标准规范、性能和价格度量，还负责管理测试结果的发布。而所有的发布结果都将由 TPC 授权的独立审核员审核。

同时，为了保证结果的公允，TPC 还成立了技术咨询委员会（TAB）。如果有人对 TPC 发布的结果有疑问，他们可以去 TAB 理论。注意，他们只会发布应用程序性能评估的基准性能测试标准规范，企业的测试代码是可以自己编写的。

那 TPC 和 TPC-A、TPC-B、TPC-C、TPC-E 有什么关系呢？我们其实可以把它们看成是 TPC 推出的基准性能测试标准规范的不同版本，TPC-A、TPC-B 是前两个版本，TPC-C 版本出来之后，就迅速替换了前两个版本。

TPC-A（1989 年发布) 主要是对借记卡在局域网和广域网中的性能进行基准评估，它增加了对 ACID 的要求。

TPC-B（1991 年发布）使用了和 TPC-A 相同的事务类型，但它取消了 TPC-A 中的网络和用户交互组件，剩下的是一个批处理事务基准。

TPC-C（1992 年发布）使用的是商品批发销售公司的模型。它主要使用了新订单、支付操作、发货、订单状态查询、库存状态查询这五个事务来执行并发，**TPC-C 测试的结果主要有两个指标，也就是每分钟事务数（tpmC）和性价比（Price/Performance，简写为 Price/tpmC）。**

而 TPC-E（2007 年发布）使用的是证券交易所的业务模型，它模拟了 12 种事务类型，包括交易查询事务、交易执行事务等。

TPC 发布 TPC-E 的初衷是想让它替代 TPC-C 的，但是从发布到现在十几年了，看起来他们并没有得到预期的效果，据说只有 SQL-Server 会刷 TPC-E 的榜单。

既然有这么多企业去刷 TPC-C 的榜单，那它的指标具体是怎么计算的呢？下面我们就来看看。

TPC-C 的 tpmC 指标计算公式一般是下面这样：

$$tpmC = \frac{日均请求量 \times 请求峰值系数 \times 请求复杂度 \times (1 + 预留扩展)}{请求时长（以小时为单位）\times 60 \times 资源最佳利用率}$$

我来简单解释一下：

日均请求量：后台请求数。

请求峰值系数：这个值从历史经验数据中得来，像有些银行就是 2-3 之间。

请求复杂度：这里需要换算一下，看看一个请求约为多少个 TPC-C 的标准请求。

预留扩展：以用户数增长、请求量增长来预估系统处理能力的增长。没有固定值，像银行系统有个 3-5 倍已经挺高的，而互联网系统就不一而足了，预估成百上千倍的增长量的大有人在，这就取决于资本如何运作了。

资源最佳利用率：通常这个数据为了好看，又为了上下都交得了差，会界定在 60%-80% 左右，没有严格的规定。

根据这个公式，我们代入一组数据来算一下：

$$tpmC = \frac{300000 \times 3 \times 10 \times (1+3)}{8 \times 60 \times 60\%} \approx 12,500,0$$

上面这些数字代表的内涵跟前面的标准公式是一一对应的，我就不单独解释了。

在标准的 TPC-C 基准中，如果我们在一个或固定的几个机器上部署了一个完整的系统，那么通过这个公式，我们就可以计算出一个指标值来。

你可能会觉得，拿到这么一组数据之后，如果我们想要知道这个值需要多少硬件资源，就可以拿计算出的值和 TPC-C 标准表来对照了。别人一个主机硬件的 TPC-C 值如果是 900，那么，你需要的就是 $\frac{12,500,000}{900} \approx 139$（台） 主机。

这个计算过程看起来非常合理，又有指标可以参考，感觉没什么难度。但是你在具体的项目中想要计算的时候，却并没有这么简单。原因主要有下面四点：

首先，你的系统的架构部署和标准的测试系统不一样。这是一个很难平衡的事情，因为每个业务系统都或多或少有一些独特设计，这些根据业务特性所做的部署架构设计会和 TPC-C 的跑分系统有差异，所以我们不能直接用 TPC-C 的跑分值来做参考。

其次，请求复杂度在大部分系统中都是比较难计算的。你可能觉得可以“使用 O(n) 复杂度表示法”，但是这种表示法在代码方法级别是容易计算的，而到了微服务分布式系统中，由于服务拆分较多导致了服务间调用开始变多，O(n) 就很难表达出架构级的复杂度了。

第三，资源最佳利用率很难匹配到生产系统。根据我的经验，在生产环境中，很多企业投入的硬件资源都远远高于所谓的最佳利用率的值。银行系统不用说，为了保证稳定性，系统资源利用率超过 15% 的时候都非常少。而互联网大厂也只有在一两个业务峰点的资源利用率会高，但为了保证系统在业务峰点能够正常运行，互联网大厂也几乎是百倍千倍地提前准备冗余的资源。

第四，峰值系数也是很难确定的。一方面，生产数据拿到的只能是历史数据，要想用这个数据来计算未来的需求，只有一种方法能让计算结果准确，那就是业务模型和业务量都不

发生大的变化，但这是几乎没法预测和保障的。另一方面，企业在发展，业务也会发生变化，这时候如果仍然用历史数据来计算 tpmC 也会不准确。

鉴于这些原因，虽然 TPC-C 的基准测试是标准的，基准测试中的数据也是准确的，但到了真实的业务系统中，这个值却只能用来参考。

如果 TPC-C 提供的逻辑不是那么精准。那怎么办呢？我们有没有一种更为精准的方式呢？下面我就来聊一聊我认为更为精准的容量评估方式：排队论。

排队论容量评估

因为排队论完整讲起来需要很多的数学理论知识，再开一个新专栏都不够讲的，所以这里我只是给你提供一个思考的方向和实际落地的 demo 实现。

什么是排队论呢？

> 排队论 (queuing theory) ，是研究系统随机聚散现象和随机服务系统工作过程的数学理论和方法，又称随机服务系统理论，为运筹学的一个分支。

排队论的模型表示为：X/Y/Z/A/B/C。其中：

> X：到达时间分布；
> Y：服务时间分布；
> Z：线程数；
> A：系统容量限制；
> B：请求源数目；
> C：服务规则。

通常我们认为一个系统的系统容量限制和请求源数目，也就是 A 和 B 都是无限大的，因为系统是可以一直提供服务的。那 C 呢，如果是请求 - 响应式的同步系统，我们可以认为是先进先出的服务规则；如果是异步情况，就可以考虑平均的服务规则。Z 是服务线程数，这个比较容易理解。但是这里我要说明一下，当我们使用排队论算一个服务时，Z 可以看作是服务中的线程数；但如果我们想用排队论来看整个微服务分布式架构的容量能力时，应该把 Z 看作是并行的服务节点个数。X 可以看作 TPS，Y 可以看成是响应时间。

这样一来，我们就可以把上面的逻辑运用在具体的项目计算过程中了。下面我们就结合 demo 代码来看一下计算逻辑。

这里我用 R 语言来实现，因为在 R 语言中有一个 queuecomputer 的开源项目，里面实现了一些基本的算法。下面我给出了实现这个 demo 的代码，代码相应的解释，我也都写到了每行的后面。

复制代码

```
library(queuecomputer)

set.seed(4)  #定义随机数种子
NumberOfRequests <- 40000000 #定义请求数
lambda_a <- 200/1  #到达率
lambda_s <- 167/1 #服务率
interarrivals <- rexp(NumberOfRequests, lambda_a) #用户或请求呈指数分布
arrivals <- cumsum(interarrivals)          #用户或请求总数
service <- rexp(NumberOfRequests, lambda_s) #用户或请求得到的服务呈指数分布

# 服务器个数（也可称为服务线程个数，取决于模型创建在哪个Leverl
NumberOfServers = 5

QeuingServerN <- queue_step(arrivals = arrivals, service = service, servers =
QeuingServerN
QeuingServerN_summary <- summary(QeuingServerN) # 计算队列的摘要数据
QeuingServerN_summary # 打印摘要数据
QeuingServerN_summary$slength_sum  # 总队列长度
QeuingServerN_summary$qlength_mean # 平均队列长度
QeuingServerN_summary$slength_mean # 平均请求长度（系统中的请求个数）
QeuingServerN_summary$mwt # 系统级平均等待时间
QeuingServerN_summary$mrt # 系统级平均响应时间
```

运行这段代码会得到下面的结果（部分内容）：

复制代码

```
.................
> QeuingServerN_summary
Total customers:
 40000000
Missed customers:
 0
Mean waiting time:
 1.28e-05
Mean response time:
 0.006
```

```
Utilization factor:
 0.239512129307182
Mean queue length:
 0.00256
Mean number of customers in system:
 1.2
> QueuingServerN_summary$slength_sum
# A tibble: 17 x 2
   queuelength    proportion
         <int>         <dbl>
 1           0 0.302
 2           1 0.361
 3           2 0.216
 4           3 0.0864
 5           4 0.0259
 6           5 0.00617
 7           6 0.00147
 8           7 0.000353
 9           8 0.0000854
10           9 0.0000213
11          10 0.00000489
12          11 0.00000126
13          12 0.000000230
14          13 0.0000000998
15          14 0.0000000831
16          15 0.0000000358
17          16 0.00000000722
> # 平均队列长度
> QueuingServerN_summary$qlength_mean
[1] 0.002556998
> # 平均请求长度（系统中的请求个数）
> QueuingServerN_summary$slength_mean
[1] 1.200118
> QueuingServerN_summary$mwt # 系统级平均等待时间
[1] 1.27839e-05
> QueuingServerN_summary$mrt # 系统级平均响应时间
[1] 0.006000121
```

从这个结果来看，我们总共有四千万客户请求，没有失败的请求。因为我们的服务率是 167，而到达率为 200，所以有一定的等待，还好这个等待时间并不长，只有 1.27839e-05，而平均的系统响应时间为 0.006000121，也就是 6 毫秒左右。从这个数据来看，支持四千万的总请求量，在到达率为 200，服务率为 167 的情况下，就需要 1.27839e-05+0.006 的总时间，把这个时间和你的业务指标相对比，就可以判断出是否满足业务需求了。

排队论这个逻辑也可以用到生产环境中，前提就是要获得请求数、服务线程个数以及请求到达的分布函数、响应时间的分布函数，这些数据都可以通过日志获得。如果你再发散一下思维，还可以把这个逻辑用做实时计算。比方说，在大压力的场景下，如果我们想把响应时间降到指标以下，可以用它来计算服务线程数。

除了对性能峰值的综合评估（上面我们说的两种方式），磁盘和网络资源也需要进行评估。

磁盘容量评估

我们先来看看磁盘的容量评估。

对于应用程序来说，本地保存日志一般都是循环的，循环策略可以根据文件个数、文件大小、保存时间进行设置，所以相对来说磁盘容量是容易计算的。

而对于数据库这样需要长期保存数据的地方来说，想要计算磁盘容量就要多加注意了。通常我们会创建一个这样的公式来进行计算：

$$磁盘容量 = 原始磁盘容量 + \sum_{r=1}^{单表}（记录长度 \times 记录数 \times 保存期限） \times 数$$

数据库膨胀因子：要根据具体的业务量进行评估。不同的业务模型，它们数据库的膨胀速度会有很大差别，而且不同的表的膨胀因子也不一样。

备份因子：数据库最怕丢失数据，所以备份是必须的，至于要备份几份就得看策略了。有很多重要的系统常常会有至少三个备份，分别存在本地机房、本城机房和远程机房。

有了这些数据之后，再计算总体需要的磁盘容量空间就比较容易了。但这个公式看似简单，如果你想把所有库的所有表都用这个公式计算一遍，其实也很不容易。

不过像我们这个专栏的示例项目，因为搭建得比大企业简单很多，所以数据上也会少很多。下面我们就来看看，我们的项目如果跑一分钟会增加多少数据。

我们先用下面两个语句分别查询数据库表空间和库空间大小。

查询 Mall 库的所有表空间大小。

复制代码

```
SELECT
    TABLE_NAME AS '表名',
    CONCAT(ROUND(TABLE_ROWS / 10000, 2), ' 万行') AS '行数',
    CONCAT(ROUND(DATA_LENGTH / (1024 * 1024), 2),
           'MB') AS '表空间',
    CONCAT(ROUND(INDEX_LENGTH / (1024 * 1024), 2),
           ' MB') AS '索引空间',
    CONCAT(ROUND((DATA_LENGTH + INDEX_LENGTH) / (1024 * 1024),
                 2),
           ' MB') AS '总空间'
FROM
    information_schema.TABLES
WHERE
    TABLE_SCHEMA = 'mall'
ORDER BY TABLE_ROWS DESC;
```

查询整库空间大小。

复制代码

```
SELECT
    TABLE_SCHEMA,
    CONCAT(TRUNCATE(SUM(data_length) / 1024 / 1024,
                2),
           ' MB') AS data_size,
    CONCAT(TRUNCATE(SUM(index_length) / 1024 / 1024,
                2),
           'MB') AS index_size
FROM
    information_schema.tables
GROUP BY TABLE_SCHEMA
ORDER BY data_length DESC;
```

运行之前：

表名	行数	表空间	索引空间	总空间
ums_member_receive_address	111.91 万行	142.70MB	26.58 MB	169.28 MB
ums_member	91.28 万行	328.94MB	101.16 MB	430.09 MB
oms_order_bak	34.67 万行	126.69MB	0.00 MB	126.69 MB
pms_sku_stock	1.93 万行	5.52MB	1.97 MB	7.48 MB
oms_order_10000	0.99 万行	4.52MB	0.00 MB	4.52 MB
ums_admin_login_log	0.02 万行	0.02MB	0.00 MB	0.02 MB
pms_product_attribute_value	0.01 万行	0.02MB	0.00 MB	0.02 MB
pms_member_price	0.01 万行	0.02MB	0.00 MB	0.02 MB
ums_role_resource_relation	0.00 万行	0.02MB	0.00 MB	0.02 MB
pms_product_category	0.00 万行	0.02MB	0.00 MB	0.02 MB
ums_role_menu_relation	0.00 万行	0.02MB	0.00 MB	0.02 MB
pms_product_attribute	0.00 万行	0.02MB	0.00 MB	0.02 MB
ums_resource	0.00 万行	0.02MB	0.00 MB	0.02 MB
pms_product	0.00 万行	0.02MB	0.00 MB	0.02 MB
pms_product_ladder	0.00 万行	0.02MB	0.00 MB	0.02 MB
pms_product_full_reduction	0.00 万行	0.02MB	0.00 MB	0.02 MB
sms_coupon_history	0.00 万行	0.02MB	0.03 MB	0.05 MB
ums_menu	0.00 万行	0.02MB	0.00 MB	0.02 MB
oms_order_operate_history	0.00 万行	0.02MB	0.00 MB	0.02 MB

TABLE_SCHEMA	data_size	index_size
mall	609.50 MB	129.78MB
performance_schema	0.00 MB	0.00MB
mysql	2.29 MB	0.19MB
information_schema	0.15 MB	0.00MB
sys	0.01 MB	0.00MB

满负载运行一分钟后：

表名	行数	表空间	索引空间	总空间
ums_member_receive_address	111.91 万行	142.70MB	26.58 MB	169.28 MB
ums_member	91.28 万行	328.94MB	101.16 MB	430.09 MB
oms_order_bak	34.67 万行	126.69MB	0.00 MB	126.69 MB
pms_sku_stock	1.93 万行	5.52MB	1.97 MB	7.48 MB
oms_order_10000	0.99 万行	4.52MB	0.00 MB	4.52 MB
oms_order	0.98 万行	3.52MB	0.39 MB	3.91 MB
ums_admin_login_log	0.02 万行	0.02MB	0.00 MB	0.02 MB
pms_product_attribute_value	0.01 万行	0.02MB	0.00 MB	0.02 MB
pms_member_price	0.01 万行	0.02MB	0.00 MB	0.02 MB
ums_role_resource_relation	0.00 万行	0.02MB	0.00 MB	0.02 MB
pms_product_category	0.00 万行	0.02MB	0.00 MB	0.02 MB
ums_role_menu_relation	0.00 万行	0.02MB	0.00 MB	0.02 MB
pms_product_attribute	0.00 万行	0.02MB	0.00 MB	0.02 MB
ums_resource	0.00 万行	0.02MB	0.00 MB	0.02 MB
pms_product	0.00 万行	0.02MB	0.00 MB	0.02 MB
pms_product_ladder	0.00 万行	0.02MB	0.00 MB	0.02 MB
pms_product_full_reduction	0.00 万行	0.02MB	0.00 MB	0.02 MB
sms_coupon_history	0.00 万行	0.02MB	0.03 MB	0.05 MB
ums_menu	0.00 万行	0.02MB	0.00 MB	0.02 MB
oms_order_operate_history	0.00 万行	0.02MB	0.00 MB	0.02 MB

TABLE_SCHEMA	data_size	index_size
mall	613.00 MB	130.15MB
performance_schema	0.00 MB	0.00MB
mysql	2.29 MB	0.19MB
information_schema	0.15 MB	0.00MB
sys	0.01 MB	0.00MB

从 Mall 库的整体大小来计算，数据库表空间一分钟增加了 3.87M。如果是一天呢，那就是近 5.4G。这只是数据库空间的增加，像日志之类的还要另算。

到这里，磁盘容量的评估就完成了。这样根据实际的测试过程计算出的结果是比较靠谱的。

网络容量评估

接下来我们还要算一下网络容量。

要想计算网络容量，首先得知道一个系统稳定运行的整体最大 TPS。当然了，要想知道这个 TPS 峰值，你得先保证我们前面讲过的容量场景已经执行通过了。

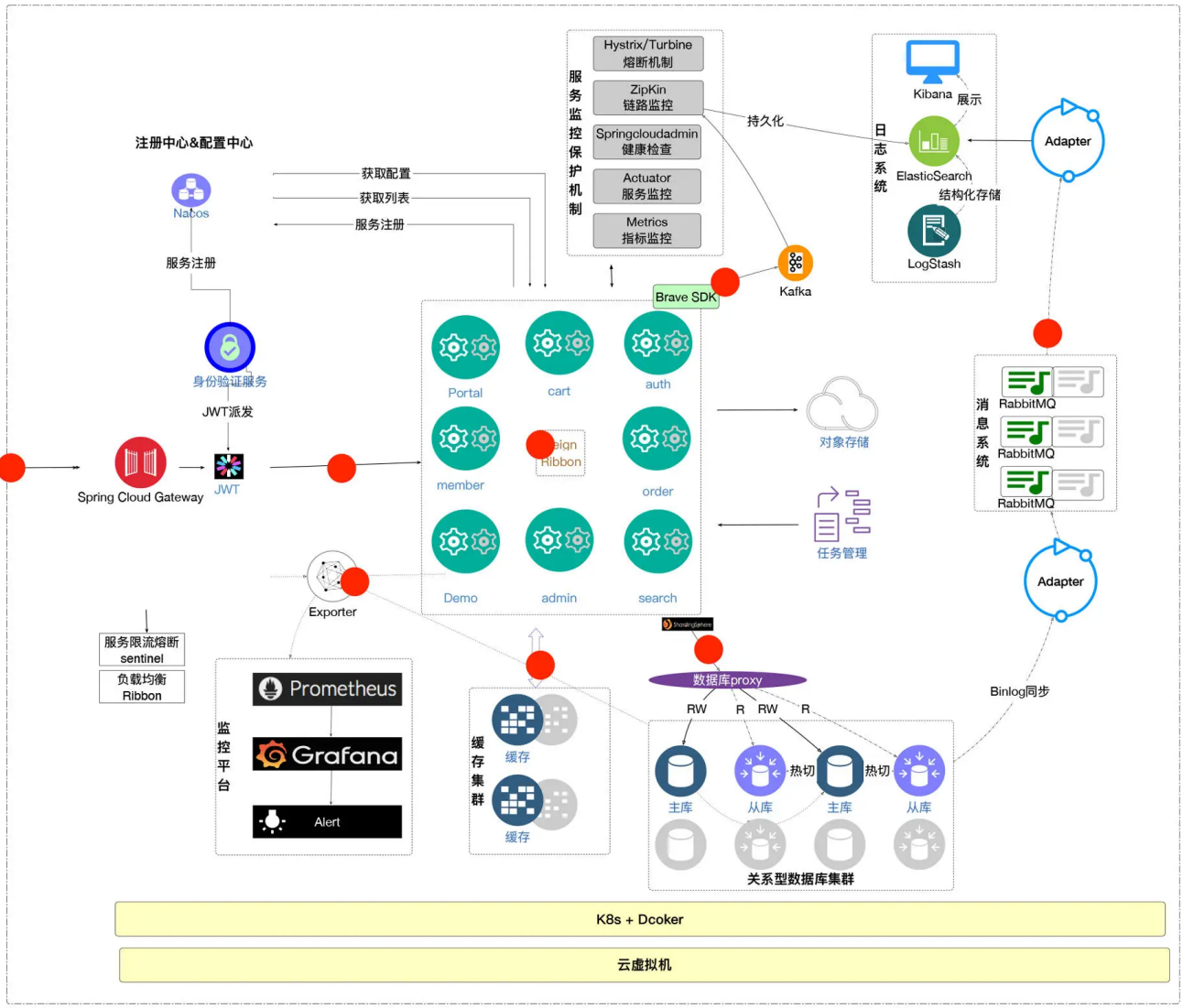
因为容量场景已经包括了像业务模型、测试数据、测试环境等等的前提条件，所以当我们得到了系统稳定运行的最大 TPS 之后，就可以直接用下面这个公式进行计算了：

$$上行带宽 = 最大TPS \times 每秒上行数据大小$$

$$下行带宽 = 最大TPS \times 每秒下行数据大小$$

为什么要分为上下行带宽呢？主要是因为在 ISP 运营商提供的网络服务中，上下行带宽的区别很大。像平时我们家里的宽带，工作人员非常自信地跟你说接入之后就能专享 100M 的带宽，实际上完全达不到这个值。

家里带宽不够最多也就是卡得你一个人不舒服，但企业级的带宽不够，那就是严重的生产事故了。我们再对着系统架构图来看一下有哪些带宽是需要计算的。



我这次放的架构图里，所有我标识为红点的地方，都是这个系统里需要计算带宽的位置。也就是说，不管是针对外网出口还是只是在内网服务之间，都是需要计算一下带宽的。

现在有一些企业又做了子网划分，这种情况下需要计算的点就更多了。好在计算公式不会有区别，有区别的只是每个位置 TPS 和上下行数据的大小，这些是要你通过容量场景统计得来的。

网络容量评估这一部分的内容就这么多，可以看出来，它的逻辑是比较清晰的。你只要找齐数据，代入公式计算就可以了。

总结

好了，这节课的内容就讲到这里。虽然刚才我很努力地没有讲一些数学理论（比如说：马尔可夫链），但是如果想要看懂这一讲的内容，还是需要一些数学基础的。

TPC-C 的本意应该是希望能够在业内提供一套标准的基准测试逻辑，通过它来指导系统性能容量规划。但是在具体的项目中，因为有太多的局限，所以我们只能拿它来做个参考。

相比较而言，如果我们使用排队论来计算容量的话会更加精准。我们的原始数据只要来自于真实的系统，并且通过检验方法（比如说卡方检验、二项检验、K-S 检验等）得到了分布函数，就可以根据排队论的逻辑进行系统的容量评估了。

请注意，这里我并没有代入前面我们提到的各种性能计数器。如果在实际的项目中，我们可以将性能计数器也作为容量规划模型的输入条件，模型将变得更加完整，数据也会更为准确。

而磁盘空间和网络容量的评估，我们直接根据实际的测试过程来做计算即可。

关于容量评估我就说到这里，希望能给你一些启发。

课后题

学完这节课，请你思考两个问题：

1. 你是如何来做系统级容量评估的？你有没有参照过具体的生产数据？
2. 你能不能用统计学中的模型（比如差分自回归移动模型 ARIMA）来代替排队论呢？

欢迎你在留言区与我交流讨论。我们下节课见！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 33 | 异常场景：如何模拟线上不同组件的异常场景？

更多学习推荐

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。